TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------

**Bài tập lớn môn học**

Giảng viên hướng dẫn: TS. Hoàng Văn Thông

Sinh viên thực hiện: Nguyễn Phan Hải

Mã sinh viên: 221230823

Lớp: CNTT 3 – K63

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**Hà Nội tháng 11 năm 2023**

# MỤC LỤC

[MỤC LỤC 2](#_Toc149906892)

[BÀI 1 (BÀI SỐ 20 TRONG DANH SÁCH BÀI TẬP) 3](#_Toc149906893)

[1.1. Đề bài 3](#_Toc149906894)

[1.2. Phân tích bài toán 3](#_Toc149906895)

[1.2. Cài đặt các lớp và hàm main bằng C++ 7](#_Toc149906896)

[1.3. Phân tích thời gian chạy của từng phương thức có trong các lớp 7](#_Toc149906897)

[BÀI 2 (BÀI SỐ 22 TRONG DANH SÁCH BÀI TẬP) 19](#_Toc149906898)

[2.1. Đề bài 19](#_Toc149906899)

[2.2. Phân tích bài toán 20](#_Toc149906900)

[2.3. Cài đặt các lớp và hàm main bằng C++ 21](#_Toc149906901)

[2.4. Phân tích thời gian chạy của từng phương thức có trong các lớp 22](#_Toc149906902)

[DANH SÁCH TÀI LIỆU THAM KHẢO 26](#_Toc149906903)

# 

# BÀI 1 (BÀI SỐ 20 TRONG DANH SÁCH BÀI TẬP)

## Đề bài

1. Xây dựng lớp Vector

2. Xây dựng quản lý một cửa hàng bán tạp hóa, thông tin quản lý của cửa hàng gồm:

+ **Danh mục mặt hàng** gồm các thông tin: Mã hàng, Tên hàng, nhà sản xuất

+ **Hóa đơn mua bán hàng**: Mã hóa đơn, Mã hàng, Loại hóa đơn (Bán hoặc Mua), Số lượng, Ngày mua bán, Giá mua bán. Với các phương thức:

a. Đọc danh thông tin quản lý của cửa hàng từ file vào chương trình

b. Ghi thông tin quản lý của cửa hàng vào file

c. Nhập xuất các mặt hàng

d. Hiển thị toàn bộ danh sách mặt hàng hiện có

e. Thống kê số lượng mặt hàng tồn của mỗi mặt hàng

2. Viết hàm main thực hiện các chức năng trên

## 1.2. Phân tích bài toán

* Yêu cầu của bài toán
* Xây dựng lớp Vector
* Ứng dụng Vector xây dựng lớp quản lí 1 cửa hàng bán tạp hóa. Trong đó có 2 lớp là lớp mặt hàng và lớp hóa đơn

+ Lớp mặt hàng gồm những thuộc tính sau: Mã hàng, Tên hàng, Nhà sản xuất

+ Lớp hóa đơn gồm những thuộc tính sau: Mã hóa đơn, Mã hàng, Loại hóa đơn (Bán hoặc Mua), Số lượng, Ngày mua bán, Giá mua bán.

* Xác định các lớp, các thuộc tính, phương thức của lớp
* Lớp vector
* Các thuộc tính
* int capacity, number;
* T \*arr;
* Các phương thức
* MyVector();
* MyVector(int k, T x);
* int getCapacity();
* int size();
* bool empty();
* void pop\_back();
* void erase(T \*it);
* void resize(int newCapacity);
* T &back();
* T &operator[](int k);
* void push\_back(T x);
* void insert(int k, T x)
* MyVector &operator=(MyVector<T> a);
* T\* begin();
* T\* end();
* T\* rbegin();
* T\* rend();
* Lớp Mặt hàng
* Các thuộc tính
* string MaHang, TenHang, NhaSanXuat;
* Các phương thức
* MatHang();
* string getMaHang();
* string getTenHang();
* string getNhaSanXuat();
* void nhap();
* void xuat();
* Lớp Hóa Đơn
* Các thuộc tính
* string MaHoaDon, NgayThang;
* int SoLuong, Gia, LoaiHoaDon;
* Các phương thức
* HoaDon();
* string getMaHoaDon();
* int getLoaiHoaDon();
* string getNgayThang();
* int getSoLuong();
* int getGia();
* void setSoLuong(int a);
* friend istream& operator>>(istream& in, HoaDon &a);
* friend ostream& operator<<(ostream& out, HoaDon &a);
* Mô tả chức năng của từng lớp, từng phương thức
* Lớp vector
* Lớp vector định nghĩa cấu trúc dữ liệu vector trừu tượng giúp người dùng quản lí mặt hàng và hóa đơn
* **Chức năng của các phương thức trong lớp**
* MyVector(): Hàm tạo không đối gán giá trị khởi đầu cho các thuộc tính.
* MyVector(int k, T x): Hàm tạo có đối cấp phát và gán k giá trị x cho mảng
* int getCapacity(): Trả về sức chứa hiện tại của vector.
* int size(): Trả về số phần tử hiện đang được lưu trong vector.
* bool empty(): Kiểm tra vector rỗng.
* void pop\_back(): Xóa phần tử cuối cùng.
* void erase(T \*it): Xóa phần tử ở một vị trí bất kì.
* void resize(int newCapacity): Cấp phát lại bộ nhớ cho mảng dữ liệu với sức chứa mới.
* T &back(): Trả về phần tử cuối của mảng.
* T &operator[](int k): Nạp chồng toán tử truy xuất dữ liệu thông qua vị trí.
* void push\_back(T x): Đẩy phần tử mới vào cuối vào vector.
* void insert(int k, T x): Chèn phần tử vào vị trí k.
* MyVector &operator=(MyVector<T> a): Nạp chồng toán tử bằng cho hai vector.
* T\* begin(): Trả về con trỏ đến đầu mảng của vector.
* T\* end(): Trả về con trỏ sau phần tử cuối của mảng.
* T\* rbegin(): Trả về con trỏ sau phần tử cuối của mảng.
* T\* rend(): Trả về con trỏ đến đầu mảng của vector.
* Lớp Mặt Hàng
* Lớp mặt hàng giúp người dùng quản lý các thông tin của một đối tượng mặt hàng.
* **Chức năng của các phương thức trong lớp**
* MatHang(): Hàm tạo không đối gán giá trị khởi đầu cho các thuộc tính.
* int getMaHang(): Trả về mã hàng.
* string getTenHang(): Trả về tên hàng.
* string getNhaSanXuat(): Trả về tên của nhà sản xuất.
* Lớp Hóa Đơn
* Giúp người dùng quản lý các đối tượng hóa đơn.
* **Chức năng của các phương thức trong lớp**
* HoaDon(): Hàm tạo không đối gán giá trị khởi đầu cho các thuộc tính.
* string getMaHoaDon(): Trả về mã hóa đơn.
* int getLoaiHoaDon(): Trả về loại hóa đơn (1.mua , 2.bán)
* string getNgayThang(): Trả về ngày tháng.
* int getSoLuong(): Trả về số lượng hàng đã mua hoặc bán.
* int getGia(): Trả về giá mua hoặc bán.
* void setSoLuong(): Thay đổi số lượng sau khi mua hoặc bán.

## Cài đặt các lớp và hàm main bằng C++

Cncio

## Phân tích thời gian chạy của từng phương thức có trong các lớp và hàm main

* Lớp vector

template <class T>

MyVector<T>::MyVector() // Gán giá trị khởi đầu cho các thuộc tính O(1).

{

this->capacity = 0;

this->number = 0;

this->arr = NULL;

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

MyVector<T>::MyVector(int k, T x)

{

this->capacity = k; // Gán giá trị cho sức chứa vector O(1).

this->number = k; // Gán giá trị cho biến biểu diễn số phần tử hiện tại trong vector O(1).

this->arr = new T[k]; // Cấp phát bộ nhớ cho k phần tử O(k).

for (int i = 0; i < k; i++) // Gán giá trị cho k phần tử O(k).

{

this->arr[i] = x;

}

}

* Phương thức có độ phức tạp tiệm cận O(k).

template <class T>

MyVector<T>::~MyVector()

{

if (arr != NULL) // So sánh giá trị arr với NULL O(1).

{

delete[] arr; // Giải phóng bộ nhớ O(capacity) capacity là sức chứa.

}

}

* Phương thức có độ phức tạp tiệm cận O(capacity).

template <class T>

int MyVector<T>::getCapacity()

{

return this->capacity; // Trả về giá trị sức chứa O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

int MyVector<T>::size()

{

return this->number; // Trả về biến lưu số phần tử hiện tại trong vector O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

bool MyVector<T>::empty()

{

return this->number == 0; // Trả về giá trị sức chứa và so sánh với 0 O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

void MyVector<T>::pop\_back()

{

if (!empty()) // Kiểm tra vector rỗng O(1)

{

this->number--; // Giảm số lượng phần tử O(1).

}

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

void MyVector<T>::resize(int newCapacity)

{

if (newCapacity < this->capacity) // Kiểm tra sức chứa mới và sức chứa hiện tại O(1).

{

return; // Nếu sức chứa mới nhỏ hơn hiện tại thì trả về O(1).

}

this->capacity = newCapacity; // Gán giá trị sức chứa mới cho sức chứa hiện tại O(1).

T \*temp = new T[this->capacity]; // Cấp phát bộ nhớ cho mảng tạm O(capacity).

for (int i = 0; i < number; i++)

{

temp[i] = arr[i]; // Gán các giá trị của mảng cũ cho mảng mới O(number).

}

if (this->arr != NULL) // Kiểm tra mảng cũ khác rỗng O(1).

{

delete[] this->arr; // Xóa mảng cũ O(capacity).

}

this->arr = temp; // Gán mảng cũ bằng mảng có sức chứa mới O(1).

}

* Phương thức có độ phức tạp tiệm cận O(capacity).

template <class T>

T &MyVector<T>::back()

{

return this->arr[number - 1]; // Trả về phần tử cuối vector O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

T &MyVector<T>::operator[](int k)

{

return this->arr[k]; // Trả về phần tử thứ k O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

void MyVector<T>::push\_back(T x)

{

if (this->number == this->capacity) // Kiểm tra giới hạn lưu trữ vector O(1).

{

resize(this->capacity \* 2 + 2); // Gọi hàm resize mở rộng vùng nhớ O(capacity \* 2 + 2).

}

arr[number] = x; // Gán giá trị vào cuối mảng O(1).

number++; // Tăng số phần tử trong vector O(1).

}

* Phương thức có độ phức tạp tiệm cận O(capacity).

template <class T>

void MyVector<T>::insert(int k, T x)

{

if (this->number == this->capacity) // Kiểm tra giới hạn lưu trữ vector O(1).

{

resize(this->capacity \* 2 + 2); // Gọi hàm resize mở rộng vùng nhớ O(capacity \* 2 + 2).

}

for (int i = this->number; i > k; i--)

{

this->arr[i] = this->arr[i - 1]; // Tiến hành dồn mảng về sau O(n – k).

}

this->arr[k] = x; // Gán giá trị vào vị trí thứ k O (1).

this->number++; // Tăng biến đếm số lượng phần tử O(1).

}

* Phương thức có độ phức tạp tiệm cận O(capacity).

template <class T>

MyVector<T> &MyVector<T>::operator=(MyVector<T> a)

{

this->capacity = a.capacity; // Gán giá trị sức chứa của vector hiện tại với sức chứa của vector được truyền vào O(1).

this->number = a.number; // Gán giá trị số phần tử của vector hiện tại với số phần tử vector được truyền vào O(1).

if (capacity) // Kiểm tra sức chứa O(1).

{

this->arr = new T[capacity]; // Cấp phát bộ nhớ cho vector hiện tại O(capacity).

for (int i = 0; i < number; i++)

{

this->arr[i] = a.arr[i]; // Gán giá trị cho mảng của vector hiện tại O(number).

}

}

else

{

this->arr = 0; // Gán giá trị mặc định cho mảng dữ liệu O(1);

}

return \*this; // Trả về vector hiện tại O(1).

}

* Phương thức có độ phức tạp tiệm cận O(capacity).

template <class T>

T \*MyVector<T>::begin()

{

return this->arr; // Trả về con trỏ đầu tiên của mảng dữ liệu O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

T \*MyVector<T>::end()

{

return this->arr + this->number; // Trả về con trỏ sau phần tử cuối cùng của mảng dữ liệu O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

T \*MyVector<T>::rbegin()

{

return this->arr + this->number; // Trả về con trỏ sau phần tử cuối cùng của mảng dữ liệu O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

T \*MyVector<T>::rend()

{

return this->arr; // Trả về con trỏ đầu tiên của mảng dữ liệu O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

template <class T>

void MyVector<T>::erase(T \*it)

{

if (number == 0) // Kiểm tra số phần tử của mảng dữ liệu O(1).

return; // Trả về nếu như mảng không có phần tử nào O(1).

for (it; it != arr + number; it++)

{

\*it = \*(it + 1); // Dồn mảng từ vị trí cuối mảng về vị trí it O(number).

}

number--; // Giảm giá trị của biến lưu số lượng phần tử còn lại O(1).

}

* Phương thức có độ phức tạp tiệm cận O(number).
* Lớp Mặt Hàng

MatHang()::MatHang()

{

// Gán các giá trị khởi đầu và xét xếp loại O(1).

MaHang=TenHang=NhaSanXuat=” ”;

}

* Phương thức có độ phức tạp tiệm cận O(1).

string MatHang::getMaHang()

{

return this->MaHang; // Trả về mã hàng O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

string MatHang::getTenHang()

{

return this->TenHang; // Trả về tên hàng O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

string MatHang::getNhaSanXuat()

{

return this->NhaSanXuat; // Trả về tên nhà sản xuất O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

void MatHang::nhap()

{

// Nhập thông tin của mã hàng, tên hàng, nhà sản xuất

cout << “\nNhap ma hang : ”;

getline(cin, MaHang);

cout << “Nhap ten hang: ”;

getline(cin, TenHang);

cout << “Nhap nha san xuat: ”;

getline(cin, NhaSanXuat);

}

* Phương thức có độ phức tạp tiệm cận O(1).

void MatHang::xuat()

{

// Xuất ra thông tin của mã hàng, tên hàng, nhà sản xuất.

cout << MaHang << “\t” << TenHang << “\t” << NhaSanXuat;

}

* Phương thức có độ phức tạp tiệm cận O(1).
* Lớp Hóa Đơn

HoaDon()::HoaDon()

{

// Gán các giá trị khởi đầu và xét xếp loại O(1).

MaHoaDon=NgayThang= “ ”;

}

* Phương thức có độ phức tạp tiệm cận O(1).

string HoaDon::getMaHoaDon()

{

return this->MaHoaDon; // Trả về mã hóa đơn O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

int HoaDon::getLoaiHoaDon()

{

return this->LoaiHoaDon; // Trả về loại hóa đơn mua hoặc bán O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

string HoaDon::getNgayThang()

{

return this->NgayThang; // Trả về ngày tháng mua bán O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

int HoaDon::getSoLuong()

{

return this->SoLuong; // Trả về loại số lượng mua hoặc bán O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

void HoaDon::setSoLuong(int a)

{

this->Soluong = a; // Thay đổi số lượng sau khi mua hoặc bán O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

int HoaDon::getGia()

{

return this->Gia; // Trả về giá mua hoặc bán O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

istream &operator>>(istream& in, HoaDon &a)

{

// Nhập vào các thông tin O(1).

getline(cin, a.MaHang);

getline(cin, a.TenHang);

getline(cin, a.NhaSanXuat);

getline(cin, a.MaHoaDon);

in >> a.LoaiHoaDon;

in.ignore(1); // xóa 1 khoảng trống O( l )

in >> a.SoLuong;

in.ignore(1);

getline(cin, a.NgayThang)

in >> a.Gia;

in.ignore(1);

return in;

}

* Phương thức có độ phức tạp tiệm cận O(1).

ostream &operator<<(ostream &out, HoaDon a)

{

// Xuất ra màn hình các thông tin O(1).

out << “Ma hang: ” << a.MaHang << endl;

out << “Ten hang: ” << a.TenHang << endl;

out << “Nha san xuat: ” << a.NhaSanXuat << endl;

out << “Ma hoa don: ” << a.MaHoaDon << endl;

out << “Loai hoa don (1.mua , 2.ban): ” << a.LoaiHoaDon << endl;

out << “So luong: ” << a.SoLuong << endl;

out << “Ngay thang: ” << a.NgayThang << endl;

out << “Gia: ” << a.Gia << endl;

return out;

}

* Phương thức có độ phức tạp tiệm cận O(1).
* Hàm Main

void Docthongtin(fstream &file , Myvector<HoaDon> &A , int n)

{

string FileName;

cout << “Nhap vao ten tep ban muon doc du lieu: “;

getline(cin, FileName);

file.open(FileName, ios::in); // Mở file input O( l )

if(file.is\_open()) // kiểm tra xem file đã mở hay chưa O( l )

{

for(int i=0; i<n; i++)

{

HoaDon a;

file >> a; // Nhập số hóa đơn vào file O( l )

A.push\_back(a); // Đẩy hóa đơn vừa đọc được vào vector O(capacity)

}

file.close(); // Đóng file O( l );

}

else {

cout << “Khong mo duoc file!” <<endl;

exit(0); //Thoát nếu như không mở được file O( l ).

}

}

* Phương thức có độ phức tạp tiệm cận O(n).

void Ghithongtin(fstream, &file , MyVector<HoaDon> A)

{

string FileName;

cout << “\nNhap vao ten tep ban muon luu du lieu: ”;

getline(cin, FileName);

file.open(FileName, ios::out); // Mở file output O( l )

if(file.is\_open())

{

for(auto x : A) // Duyệt qua danh sách hóa đơn O( n )

file << x << endl;

file.close(); // Đóng file O( l )

}

}

else{

cout << “\nKhong mo dc file! ” <<endl;

exit(0); //Thoát nếu như không mở được file O( l ).

}

* Phương thức có độ phức tạp tiệm cận O(n).

void nhapMatHang(MyVector<MatHang> &A, int m)

{

for(int i=0; i<m; i++)

{

MatHang a;

a.nhap(); // Nhập thông tin của các mặt hàng O( l )

A.push\_back(a); // Đẩy mặt hàng vừa nhập vào vector O(capacity)

}

for(auto x : A) // Duyệt qua danh sách mặt hàng O(n)

{

x.xuat(); // Xuất thông tin của các mặt hàng O( l )

cout <<endl;

}

}

* Phương thức có độ phức tạp tiệm cận O(n).

void HienThiDs(MyVector<MatHang> A, int m){

for(auto x : A) // Duyệt qua danh sách mặt hàng O(n)

{

x.xuat(); // Xuất thông tin của các mặt hàng O( l )

cout <<endl;

}

}

* Phương thức có độ phức tạp tiệm cận O(n).

// loaihd = 1 +

// loaihd = 2 –

void Thongke(MyVector<HoaDon> a)

{

for( int i = 0 ; i < a.size() ; i++ ){ // Duyệt các phần tử trong vector O(n)

for( int j = 0 ; j < a.size() ; j++ ){

if( a[i].getTenHang() > a[j].getTenHang() ) // so sánh tên hàng O(n)

swap( a[i] , a[j] ); // Đổi chỗ vị trí 2 phần tử trong vector O(n)

}

}

MyVector<HoaDon> m; // tạo 1 vector m rỗng O( l )

m.push\_back( a[0] ); // đẩy phẩn tử đầu tiên vào vector O(capacity)

for( int i = 1 ; i < a.size() ; i++ ){ //Duyệt các phần tử trong vector O(n)

if( a[i]).getTenHang() == m.back.getTenHang()) /

// so sánh tên hàng ở ngoài với bên trong vector O(n)

{

int t = m.back.getSoLuong(); // gán số lượng cho t O( l )

if( a[i].getLoaiHoaDon == 1){ // so sánh loại hóa đơn với 1 O(n)

t += a[i].getSoLuong;

}

else if( a[i].getLoaiHoaDon() == 2){ // so sánh loại hóa đơn với 2 O(n)

t-=a[i].getSoLuong();

}

m.back.setSoLuong(t); // Cập nhật số lượng sau khi tính toán O(n)

}

else{

m.push\_back(a[i]); // nếu đã hết tên hàng trùng nhau thì đẩy tiếp cái tên hàng tiếp theo vào vector O(capacity)

}

}

for( int i = 0 ; i < m.size() ; i++ ){

for( int j = 0 ; j < m.size() ; j++ ){

if( m[i].getTenHang() < m[j].getTenHang() )

swap( m[i] , m[j] );

}

}

for(auto x : m){ // Duyệt danh sách hóa đơn trong vector m O(n)

cout << x.getTenHang() << “So luong con lai la: ” << x.getSoLuong() << endl;

}

}

* Phương thức có độ phức tạp tiệm cận O().

# BÀI 2 (BÀI SỐ 22 TRONG DANH SÁCH BÀI TẬP)

## 2.1. Đề bài

1. Cho cây nhị phân trừu tượng (cây heap) chứa các phần tử so sánh được với nhau thỏa mãn các tính chất sau:

a. Phần tử gốc lớn hơn hoặc bằng mọi phần tử còn lại của cây

b. 2 cây con trái và phải cũng thỏa mãn tính chất trên

2. Hãy cài đặt cấu trúc dữ liệu cây nhị phân trên có các thao tác push(x) thêm phần tử vào cây và pop() lấy phần tử gốc khỏi cây, top() lấy giá trị của nút gốc sao cho luôn luôn đảm bảo tính chất cây heap Áp dụng cấu trúc cây này như một hàng đợi ưu tiên thực hiện các việc sau:

- Nhập vào 1 dãy số xuất ra dãy giảm dần

- Giải quyết bài toán nối thanh kim loại <http://laptrinhonline.club/problem/Tèonoidam>

Công việc cơ khí thật là mệt nhọc, muốn nối một thanh kim loại độ dài a với một thanh kim loại độ dài b thì kinh phí để thuê nối tốn mất a+b đơn vị tiền tệ. Hiện nay Tèo cần nối n thanh kim loại lần lượng có độ dài là a1, a2, ... an thành một đoạn theo bạn Tèo nên bố trí thế nào để tổng số tiền phải trả là ít nhất

Input:

Dòng đầu chứa số nguyên dương n (1<=n<=105 )

Dòng tiếp theo là n số nguyên dương tương ứng là độ dài các thanh muốn nối (1<=ai<=103 )

Output: Một số nguyên dương là số kinh phí ít nhất phải trả

Ví dụ

Input

3

8

4

6

Output 28

Giải thích: Nếu ta nối thanh 8 với thanh 4 tốn chi phí là 8+4=12 sau khi nối xong còn 2 thanh độ dài 12 và 6 nối lại với nhau tốn 12+6=18 tổng chi phí nối là 12+18=30. Nếu ta nối 4 với 6 trước tốn 10 và còn 2 thanh 10 và 8 nối lại với nhau tốn 18 do đó tổng kinh phí ít hơn chỉ còn 28 Gợi ý: Bằng cách nhập vào dãy a là độ dài thanh mỗi bước lấy ra 2 thanh ngắn nhất nối với nhau rồi lại thêm thanh nối được vào

## 2.2. Phân tích bài toán

* Yêu cầu của bài toán
* Xây dựng cây nhị phân trừu tượng (cây heap)
* Cài đặt cấu trúc dữ liệu cây nhị phân trên và ứng dụng cây đó giải quyết bài toán nối thanh kim loại trên http://laptrinhonline.club/problem/Tèonoidam
* Xác định các lớp, các thuộc tính, phương thức của lớp
* Cây Heap
* Các thuộc tính
* int \*a;
* int sz;
* Các phương thức
* Heap();
* ~Heap();
* int getsize();
* void heapfy\_arr(int \*a, int n, int i)
* void push(int value);
* void pop();
* int top();
* void xuat();
* void heapsort();
* Mô tả chức năng của từng lớp, từng phương thức
* Cây heap
* Cây heap định nghĩa các thao tác như thêm phần tử vào cây, lấy phần tử gốc khỏi cây, …
* **Chức năng của các phương thức trong lớp**
* Heap(): Hàm tạo không đối gán giá trị khởi đầu cho các thuộc tính.
* ~Heap(): Giải phóng bộ nhớ
* int getsize(): Trả về kích thước mảng.
* void heapify\_arr(int \*a, int n, int i): Xây dựng heap nhị phân từ mảng ( min heap ).
* void push(int value): Thêm 1 phần tử vào cây
* void pop(): Lấy phần tử gốc ra khỏi cây
* int top(): Lấy giá trị của nút gốc của cây.
* void xuat(): In ra các phần tử của cây.
* void heapsort(): sắp xếp các phần tử trong cây (sắp xếp vun đống)

## 2.3. Cài đặt các lớp và hàm main bằng C++

[Link to code on Github](https://github.com/vinhtruong204/DSA-UTC)

## 2.4. Phân tích thời gian chạy của từng phương thức có trong các lớp

* Cây Heap

Heap()

{

a = new int[100000];

sz = 0;

}

* Phương thức có độ phức tạp tiệm cận O(1).

~Heap()

{

delete[] arr; // Giải phóng bộ nhớ O(capacity) capacity là sức chứa.

a = NULL;

}

* Phương thức có độ phức tạp tiệm cận O(capacity).

int getsize()

{

return sz; // Trả về kích thước mảng O(1).

}

* Phương thức có độ phức tạp tiệm cận O(1).

void heapify\_arr(int \*a, int n, int i)

{

if( n == 1){ // so sánh n (n là số phần tử trong mảng) với 1 O( l )

return ; // mảng có 1 phần tử thì không cần cập nhật

}

else

{

int largest = i; // lưu vị trí nút max ban đầu

int left = 2\*i + 1; // vị trí nút trái

int right = 2\*i +2; // vị trí nút phải

if(left < n && a[left] < a[largest]){

// nếu tồn tại nút left nhỏ hơn nút cha, gán min = left

largest = left;

}

if(right < n && a[right] < a[largest]){

// nếu tồn tại nút right nhỏ hơn nút cha, gán min = right

largest = right;

}

if(largest != i){ // nếu nút cha không phải là nhỏ nhất

swap(a[i] , a[largest]); // đổi chỗ 2 phần tử nếu giá trị của nút cha lớn hơn nút con

heapify\_arr(a,n,largest); // đệ quy nút tại vị trí mới

}

}

}

* Phương thức có độ phức tạp tiệm cận O(n\*log n).

void push(int value)

{

if(sz == 0){

// so sánh kích thước mảng với 0, nếu rỗng thì thêm ptu value và kích thước +1

a[sz] = value;

sz+=1;

}

else{

a[sz] = value;

// nếu khác không thì thêm ptu value, kích thước +1 và dùng đệ quy để giữ cấu trúc heap

sz+=1;

for(int i = sz/2-1; i >= 0; i++)

// duyệt các phần tử trong mảng rồi đưa phần tử vào đúng vị trí của nó

{

heapify\_arr(a,sz,i); // đệ quy nút tại vị trí mới

}

}

}

* Phương thức có độ phức tạp tiệm cận O(n\*log n).

void pop()

{

swap(a[0], a[sz -1]); // đổi vị trí của phần tử gốc với phần tử cuối cùng trong mảng

sz -=1; // giảm kích thước mảng

for(int i = sz/2-1 ; i >= 0 ; i-- )

{

heapify\_arr(a,sz,i); // dùng đệ quy để vẫn giữ đc cấu trúc heap

}

}

* Phương thức có độ phức tạp tiệm cận O(n\*log n).

int top()

{

return a[0]; // trả về giá trị a[0];

}

* Phương thức có độ phức tạp tiệm cận O( l ).

void xuat()

{

if(sz==0) // so sánh kích thước với 0

{

cout << "cay rong! \n"; // nếu bằng 0 thì in ra và trả về

return;

}

for(int i=0 ; i<sz ; i++) // duyệt qua các ptu và in ra giá trị các ptu

{

cout << a[i]<< " ";

}

cout << endl;

}

* Phương thức có độ phức tạp tiệm cận O(n).

void heapsort()

{

for(int i=sz-1; i>=0 ; i--) // duyệt các ptu bắt đầu từ ptu cuối

{

swap(a[i], a[0]); // đổi chỗ phần tử cuối với phần tử gốc

heapify\_arr(a,i,0); // đệ quy tại vị trí mới để giữ cấu trúc heap

}

}

* Phương thức có độ phức tạp tiệm cận O(n \* log n).
* Hàm main

Heap \*a = new Heap; // tạo 1 cây heap rỗng

int n;

cout << “Nhap So Phan Tu Can Them Vao Cay: ”;

cin >> n;

for(int i=0 ; i<n ; i++) // duyệt các phần tử O( l )

{

cout << “Nhap Phan Tu Thu ” << i+1 << “ : ”;

int x;

cin >>x;

a->push(x); // thêm những phần tử vừa nhập vào cây

}

* Phương thức có độ phức tạp tiệm cận O(log n).

a->heapsort() ; // sắp xếp các phần tử theo thứ tự giảm dần O(n\*log n)

cout << “ Phan Tu Duoc Sap Xep Theo Gia Tri Giam Dan Cua Mang La: \n ”;

a->xuat();

// Ứng dụng cây heap giải quyết bài thanh kim loại

cout << “Bai Tap Ve Noi Thanh Kim Loai: \n ”;

int k;

cout << “Nhap So Phan Tu: ”;

cin >> k;

Heap \*b = new Heap ; // tạo 1 cây heap rỗng mới

for(int i = 0 ; i<k ; i++ )

{

int x;

cout << “Nhap Do Dai Thanh Kim Loai Thu ” << i+1 << “ : ”;

cin >>x;

b->push(x); // thêm các ptu vừa nhập vào cây O(log n)

}

int chiphi=0;

while (b->getsize!=0) // Dùng lặp qua các ptu trong heap

{

int p = b->top(); // p là ptu nhỏ nhất đầu tiên O( 1 )

b->pop(); // lấy ra min1 O(log n)

int q = b->top() ; // q là ptu nhỏ nhất thứ 2 O( 1 )

b->pop() ; // lấy ra min2 O(log n)

b->push(p+q) ; // cộng 2 ptu đó rồi thêm vào cây và gán vào biến chi phí O(log n)

chiphi+=(p+q) ;

if(b->getsize()==1) break ;

}

cout << “Chi Phi Re Nhat La: ”<<chiphi << “\n”;

# DANH SÁCH TÀI LIỆU THAM KHẢO

1. Slide bài giảng trong trang web [lms.utc.edu.vn](https://lms.utc.edu.vn/) – Ngày tham khảo 30/10/2023.
2. Video bài giảng trong trang web [lms.utc.edu.vn](https://lms.utc.edu.vn/) – Ngày tham khảo 30/10/2023.
3. Tìm hiểu trên trang web <https://www.geeksforgeeks.org/building-heap-from-array/> – Ngày tham khảo 5/11/2023.